DAA Pract 4

Name: Avni Sameer Dongre
Roll no: A4_B3_46

Aim: Implement maximum sum of subarray for the given scenario of resource allocation using the divide and conquer approach.

Problem Statement:

A project requires allocating resources to various tasks over a period of time. Each task requires a certain amount of resources, and you want to maximize the overall efficiency of resource usage. You're given an array resources where resources[i] represents the amount of resources required for the ith task. Your goal is to find the contiguous subarray of tasks that maximizes the total resources utilized without exceeding a given resource constraint. Handle cases where the total resources exceed the constraint by adjusting the subarray window accordingly. Your implementation should handle various cases, including scenarios where there's no feasible subarray given the constraint and scenarios where multiple subarrays yield the same maximum resource utilization.

Code:

```c
#include <stdio.h>
#include <limits.h>

void printSubarray(int arr[], int start, int end) {
    printf("{ ");
    for (int i = start; i <= end; i++) {
```

```
        printf("%d ", arr[i]);
    }
    printf("}");
}
int max(int a, int b) {
    return (a > b) ? a : b;
}

int max_of_three(int a, int b, int c) {
    return max(max(a, b), c);
}

int maxCrossingSum(int arr[], int low, int mid, int high) {
    int sum = 0;
    int left_sum = INT_MIN;
    for (int i = mid; i >= low; i--) {
        sum = sum + arr[i];
        if (sum > left_sum) {
            left_sum = sum;
        }
    }

    sum = 0;
    int right_sum = INT_MIN;
    for (int i = mid + 1; i <= high; i++) {
        sum = sum + arr[i];
        if (sum > right_sum) {
            right_sum = sum;
        }
    }

    return max_of_three(left_sum + right_sum, left_sum, right_sum);
}

int maxSumRecursive(int arr[], int low, int high) {
    if (low == high) {
        printf("Individual element found: %d\n", arr[low]);
        return arr[low];
    }

    int mid = low + (high - low) / 2;

    printf("\n--- Dividing array segment ");
    printSubarray(arr, low, high);
    printf(" ---\n");
    printf("First subarray: ");
    printSubarray(arr, low, mid);
    printf("\nSecond subarray: ");
    printSubarray(arr, mid + 1, high);
    printf("\n");

    int left_max_sum = maxSumRecursive(arr, low, mid);
```

```
    int right_max_sum = maxSumRecursive(arr, mid + 1, high);

    int cross_max_sum = maxCrossingSum(arr, low, mid, high);

    printf("\n--- Applying Concept for segment ");
    printSubarray(arr, low, high);
    printf(" ---\n");
    printf("Max sum in left half: %d\n", left_max_sum);
    printf("Max sum in right half: %d\n", right_max_sum);
    printf("Max sum for crossing subarray: %d\n", cross_max_sum);

    int result = max_of_three(left_max_sum, right_max_sum, cross_max_sum);
    printf("=> Maximum of (%d, %d, %d) is %d\n", left_max_sum, right_max_sum,
cross_max_sum, result);

    return result;
}

int main() {
    int originalArray[] = {1, 2, 3, 4, 5, 6, 7};
    int size = sizeof(originalArray) / sizeof(originalArray[0]);

    printf("Applying Maximum Sum Subarray using Divide and Conquer:\n");
    printf("Original Array: ");
    printSubarray(originalArray, 0, size - 1);
    printf("\n");

    int maxSum = maxSumRecursive(originalArray, 0, size - 1);

    printf("\n----------------------------------------------------------------------\n");
    printf("Final Maximum Sum of a Contiguous Subarray is: %d\n", maxSum);
    printf("----------------------------------------------------------------------\n");

    return 0;
}
```

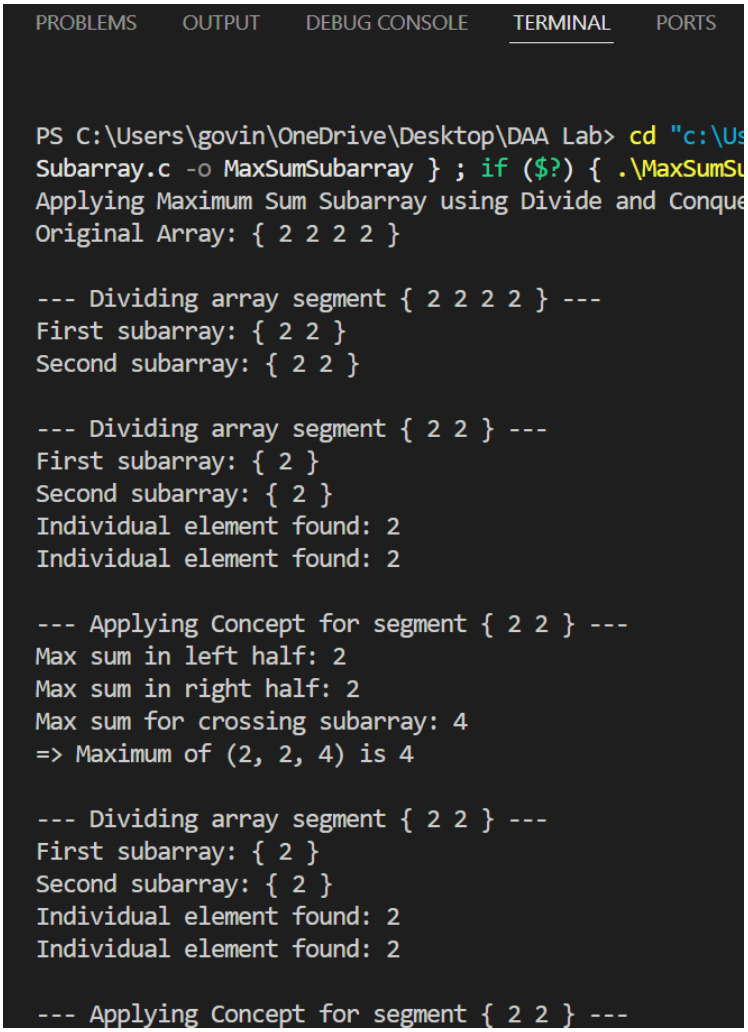
Output:

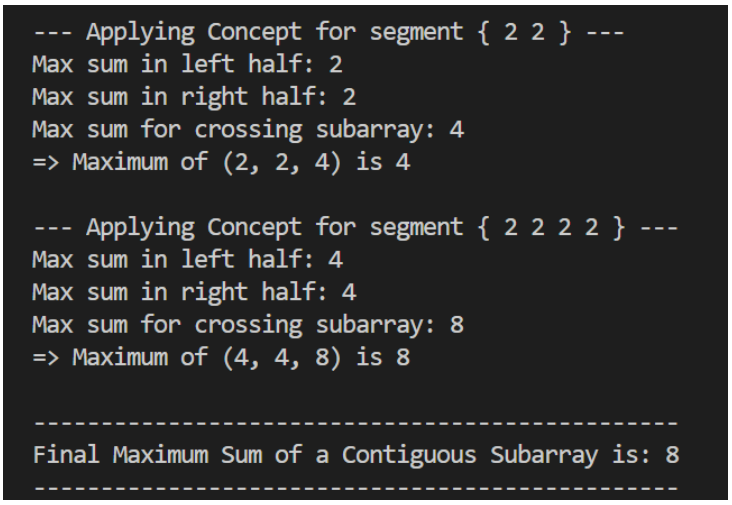
Task 1: basic array [2,1,3,4]:

```
PS C:\Users\govin\OneDrive\Desktop\DAA Lab> cd "c:\User
Subarray.c -o MaxSumSubarray } ; if ($?) { .\MaxSumSuba
Applying Maximum Sum Subarray using Divide and Conquer
Original Array: { 2 1 3 4 }

--- Dividing array segment { 2 1 3 4 } ---
First subarray: { 2 1 }
Second subarray: { 3 4 }

--- Dividing array segment { 2 1 } ---
First subarray: { 2 }
Second subarray: { 1 }
Individual element found: 2
Individual element found: 1

--- Applying Concept for segment { 2 1 } ---
Max sum in left half: 2
Max sum in right half: 1
Max sum for crossing subarray: 3
=> Maximum of (2, 1, 3) is 3

--- Dividing array segment { 3 4 } ---
First subarray: { 3 }
Second subarray: { 4 }
Individual element found: 3
Individual element found: 4

--- Applying Concept for segment { 3 4 } ---
```

```
--- Applying Concept for segment { 3 4 } ---
Max sum in left half: 3
Max sum in right half: 4
Max sum for crossing subarray: 7
=> Maximum of (3, 4, 7) is 7

--- Applying Concept for segment { 2 1 3 4 } ---
Max sum in left half: 3
Max sum in right half: 7
Max sum for crossing subarray: 10
=> Maximum of (3, 7, 10) is 10

------------------------------------------------
Final Maximum Sum of a Contiguous Subarray is: 10
------------------------------------------------
```

Task 2: same input [2,2,2,2]:





Task 3: Empty array []:

```
First subarray: { 0 }
Second subarray: { }
Individual element found: 0
```

Solved Example: [2 , -4 , 3 , -1 , 5 , -6]

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\govin\OneDrive\Desktop\DAA Lab> cd "c:\U
Subarray.c -o MaxSumSubarray } ; if ($?) { .\MaxSumS
Applying Maximum Sum Subarray using Divide and Conqu
Original Array: { 2 -4 3 -1 5 -6 }

--- Dividing array segment { 2 -4 3 -1 5 -6 } ---
First subarray: { 2 -4 3 }
Second subarray: { -1 5 -6 }

--- Dividing array segment { 2 -4 3 } ---
First subarray: { 2 -4 }
Second subarray: { 3 }

--- Dividing array segment { 2 -4 } ---
First subarray: { 2 }
Second subarray: { -4 }
Individual element found: 2
Individual element found: -4

--- Applying Concept for segment { 2 -4 } ---
Max sum in left half: 2
Max sum in right half: -4
Max sum for crossing subarray: 2
=> Maximum of (2, -4, 2) is 2
Individual element found: 3

--- Applying Concept for segment { 2 -4 3 } ---
Max sum in left half: 2
```

```
--- Applying Concept for segment { 2 -4 3 } ---
Max sum in left half: 2
Max sum in right half: 3
Max sum for crossing subarray: 3
=> Maximum of (2, 3, 3) is 3

--- Dividing array segment { -1 5 -6 } ---
First subarray: { -1 5 }
Second subarray: { -6 }

--- Dividing array segment { -1 5 } ---
First subarray: { -1 }
Second subarray: { 5 }
Individual element found: -1
Individual element found: 5

--- Applying Concept for segment { -1 5 } ---
Max sum in left half: -1
Max sum in right half: 5
Max sum for crossing subarray: 5
=> Maximum of (-1, 5, 5) is 5
Individual element found: -6

--- Applying Concept for segment { -1 5 -6 } ---
Max sum in left half: 5
Max sum in right half: -6
Max sum for crossing subarray: 5
=> Maximum of (5, -6, 5) is 5
```

```
--- Applying Concept for segment { 2 -4 3 -1 5 -6 } ---
Max sum in left half: 3
Max sum in right half: 5
Max sum for crossing subarray: 7
=> Maximum of (3, 5, 7) is 7

------------------------------------------------
Final Maximum Sum of a Contiguous Subarray is: 7
------------------------------------------------
```